NCED Summer Institute 2017

Introduction to programming bootcamp - Version Control

Acknowledgements:

This lesson borrows heavily from the following sources:

An Introduction to Version Control Using GitHub Desktop | Programming Historian

Getting your project on GitHub · GitHub Guides

Software Carpentry: Version Control with Git

GitHub Guides

Lesson objectives:

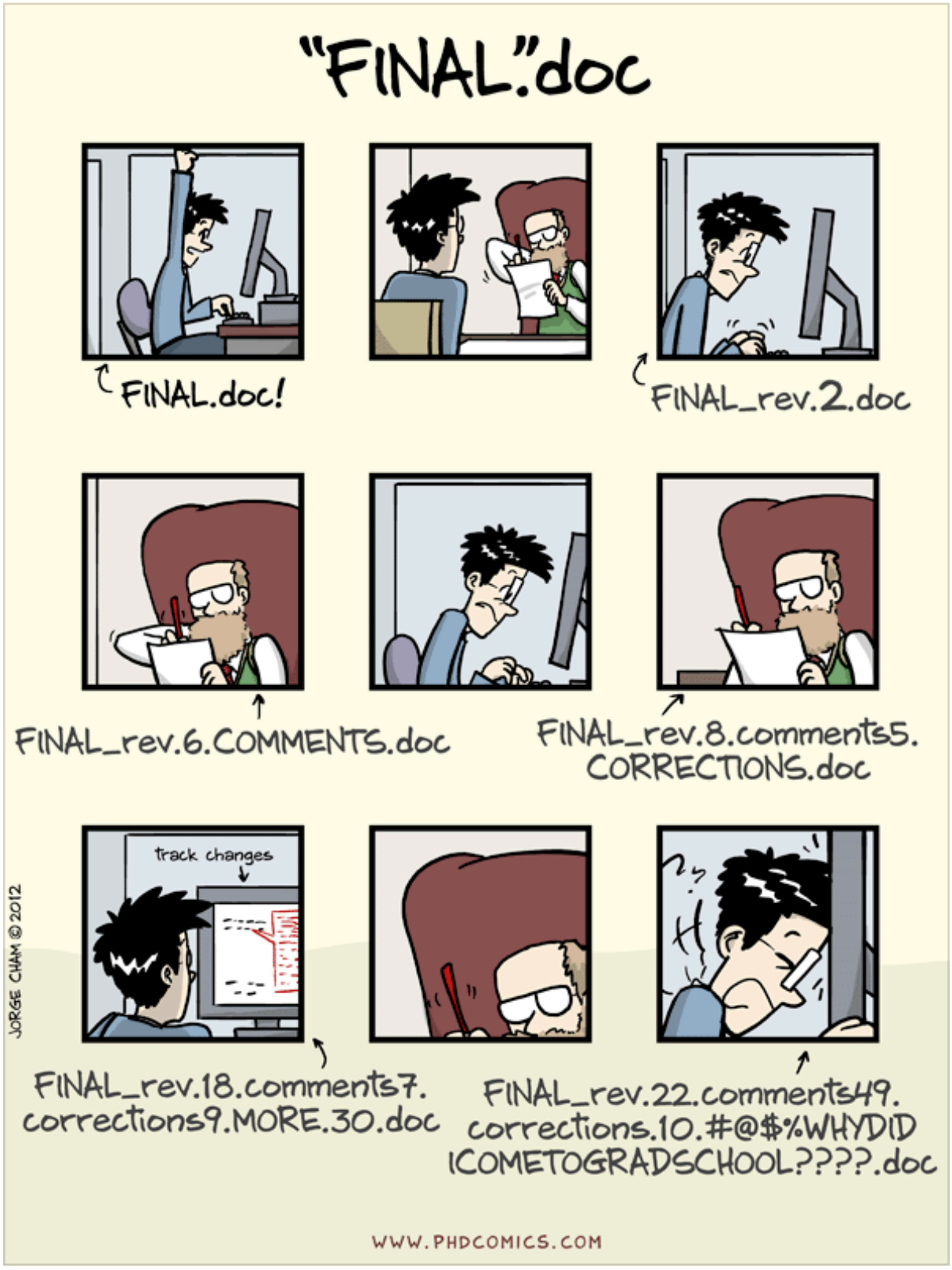
In this lesson you will be introduced to the basics of version control, understand why it is useful and implement basic version control for a plain text document using GitHub Desktop. By the end of this lesson you should understand:

- What version control is and why it can be useful
- The differences between Git and GitHub
- How to implement version control using GitHub Desktop, a Graphical User Interface for GitHub
- Other resources that will help you implement version control in your academic writing and software development

---

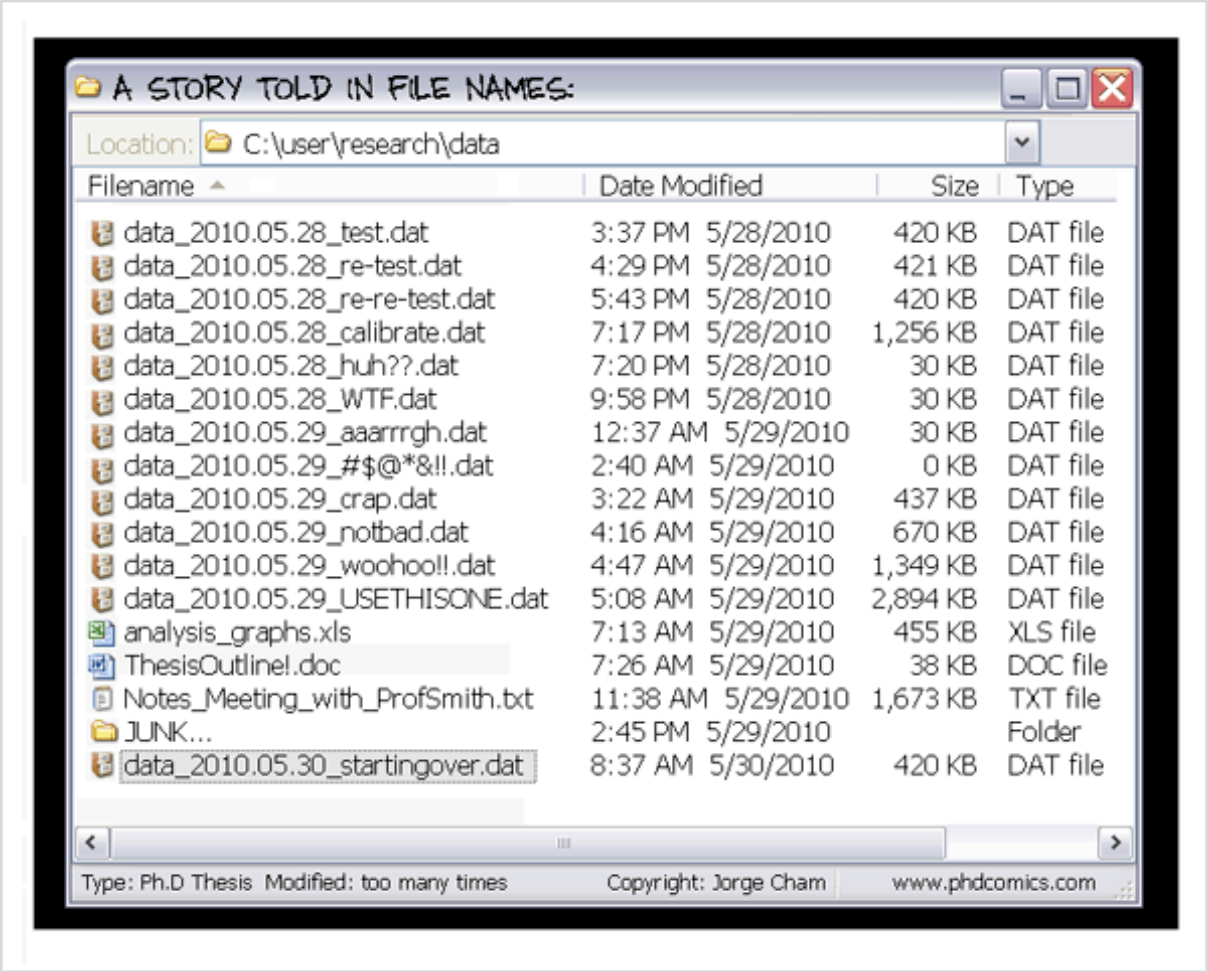
What is Version Control?

We have all been in this situation:



PHD Comics: notFinal.doc

Passing documents back and forth often leads to a mess of files that will be difficult to sort through in the future:



PHD Comics: A story in file names

Version control involves taking 'snapshots' of files at different points in time. Each snapshot records information about when the snapshot was made but also about what changes occurred between different snapshots. Many people use some sort of version control by saving different versions of the files, but it is a bit ridiculous to have multiple nearly-identical copies of the same document. Even if we carefully follow a naming convention for the files, this system doesn't record or describe the specific changes that took place between any two copies. If you have a change of heart about some of these changes, you would need to dig through the files to figure out which one has the particular version you want in order to go back to it.

Why use Version Control?

As research increasingly makes use of digital tools and storage, it becomes important to consider how to best manage our research data. This becomes especially important when we want to collaborate with other people. Though version control was originally designed for dealing with code, there are many benefits to using it to with text documents. Version controlling your files allows you to:

- Track developments and changes in your documents
- Permanently record the changes you made to your document in a way that you will be able to understand later
- Experiment with different versions of a document while maintaining the original version
- 'Merge' two versions of a document and manage conflicts between versions
- Revert changes, moving 'backwards' through your history to previous versions of your document
- Keep a record of who made what changes when.
- When several people collaborate on the same project, it's possible to accidentally overlook or overwrite someone's changes. A version control system automatically identifies conflict between one person's work and another's.

What are Git and GitHub?

Though often used synonymously, Git and GitHub are two different things. **Git** is a particular implementation of version control originally designed to manage the Linux source code. Other systems of version control exist though they are used less frequently. Git can be used to refer both to a particular approach taken to version control and the software underlying it.

GitHub is a company which hosts Git repositories and provides software for using Git like **GitHub.com** and **GitHub Desktop**. GitHub is currently the most popular host of open source projects by number of projects and number of users.

Although GitHub's focus is primarily on source code, other projects, such as The Programming Historian, are increasingly making use of version control systems like GitHub to manage the work-flows of journal publishing, open textbooks and other humanities projects.

Why not use Dropbox or Google Drive?

Dropbox, Google Drive and other services offer some form of version control in their systems. There are times when this may be sufficient for your needs. However there are a number of advantages to using a version control system like Git:

- *Language support*: Git supports both text and programming languages. As research moves to include more digital techniques and tools, it becomes increasingly important to have a way of managing and sharing both the 'traditional' outputs (journal articles, books, etc.) but also these newer outputs (code, datasets etc.)
- *More control*: A proper version control systems gives you a much greater deal of control

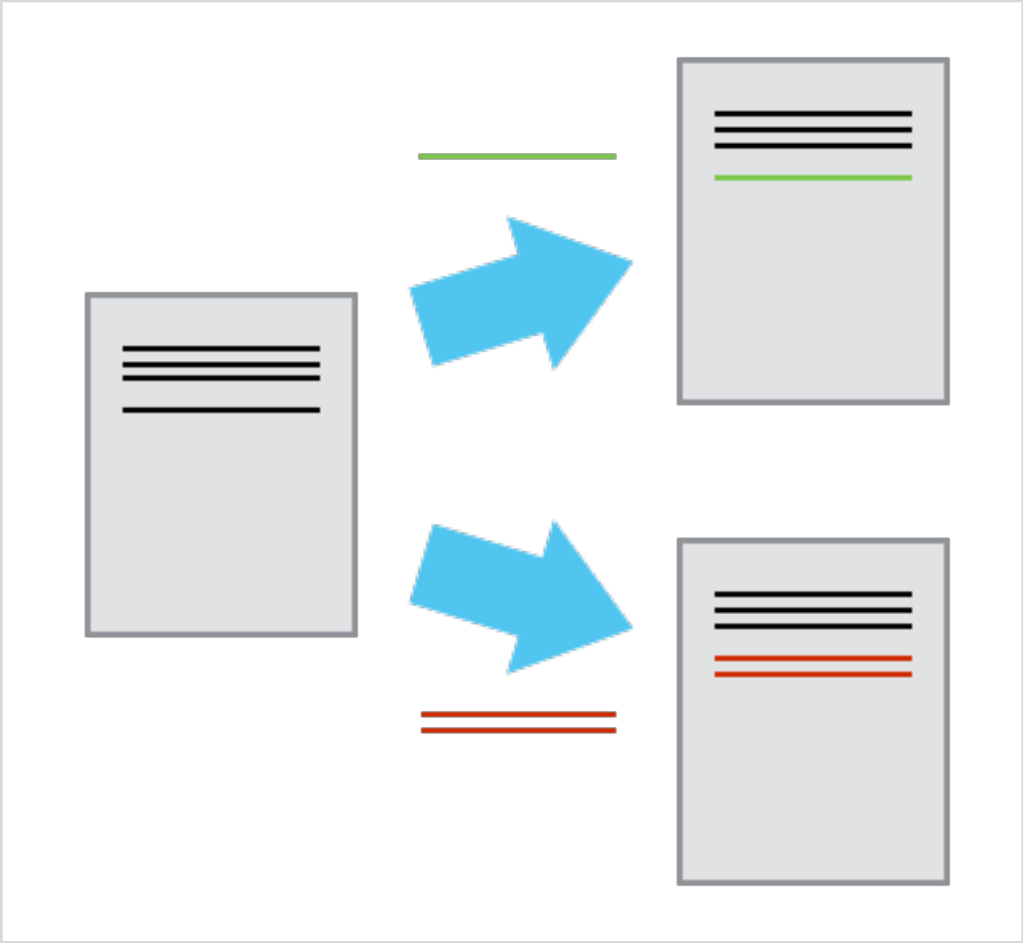
over how you manage changes in a document.

- *Useful history*: Using version control systems like Git will allow you to produce a history of your document in which different stages of the documents can be navigated easily both by yourself and by others.

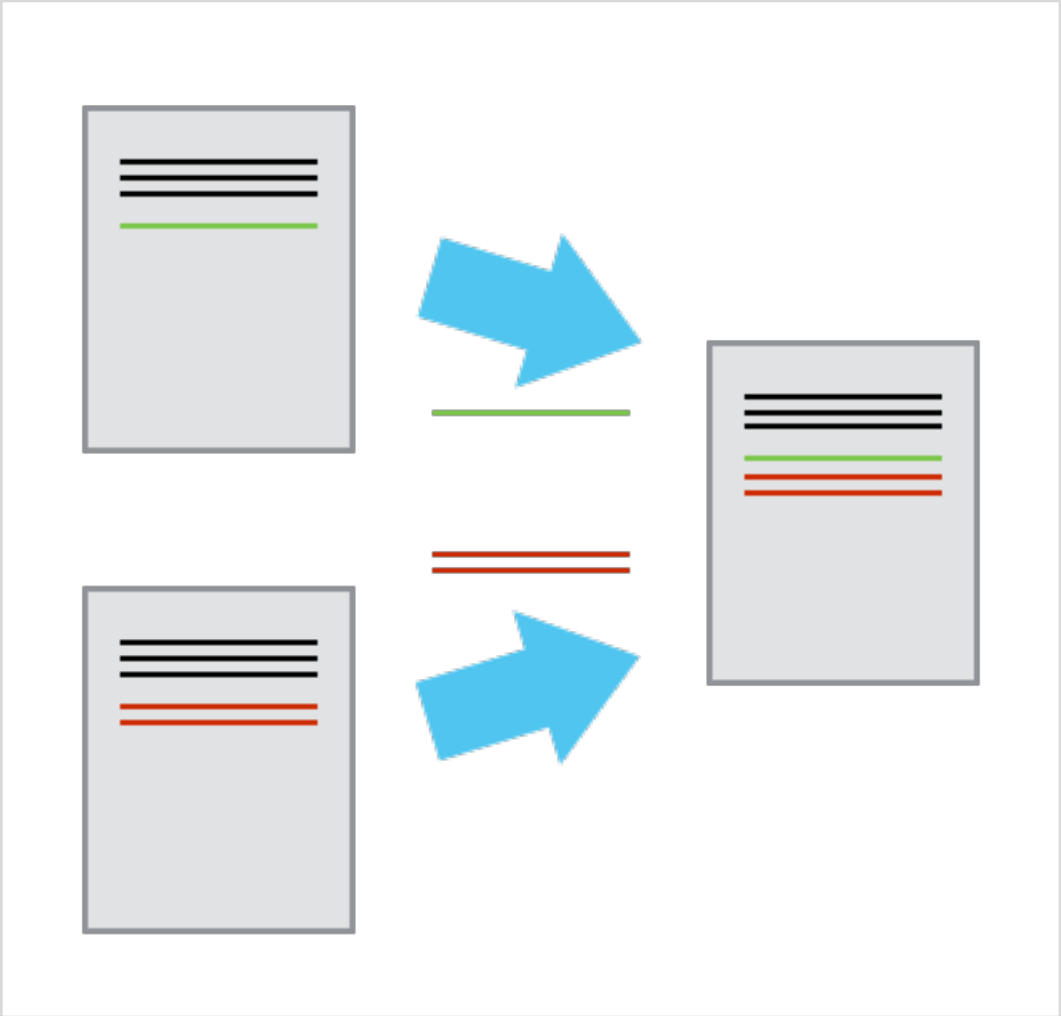
How version control works

Version control systems start with a base version of the document and then save just the differences between one version of the file and the next. You can think of it as a tape: if you rewind the tape and start at the base document, then you can play back each set of changes sequentially and end up with the latest version of the document.

Changes are separate from the document itself. Different sets of changes can turn the base document into different final versions of the document. For example, two users can make independent sets of changes based on the same document.



If there aren't conflicts, you can even try to play two sets of changes onto the same base document.



A version control system is a tool that keeps track of these changes for us and helps us version and merge our files. It allows you to decide which changes make up the next version, called a **commit**, and keeps useful metadata about them. The complete history of commits for a particular project and their metadata make up a **repository**. Repositories can be kept in sync across different computers facilitating collaboration among different people.

A Note on Terminology

One of the trickiest aspects of using Git and GitHub is the new terminology. This glossary might be useful. But in general it can be best to pick up terminology by using GitHub rather than trying to understand all of the terms before you begin using it.

xkcd: Git

Getting Started

Purists insist on using Git *only* through the command line. However, GitHub Desktop is a powerful tool that will allow us to easily start using version control. GitHub Desktop offers a Graphical User Interface (GUI) for Git. Though there are some advantages to using the command line version of Git in the long run, using a GUI can reduce the learning curve of using version control and Git. If you decide you are interested in using Git through the command line, you can find more lessons in the CSDMS education repository.

1. Register for a GitHub Account

Since we are going to be using GitHub, you will need to register for an account at GitHub.com if we don't already have one. For students and researchers GitHub offers free private repositories. These are not necessary but might be appealing if you want to keep some work private.

2. Install GitHub Desktop

The process for installing software will be slightly different depending on whether you are on Windows or Mac. Follow the instructions on GitHub's install page for GitHub Desktop.

Version Controlling a Plain Text Document

Version control systems like Git work best with **plain text** files. Plain text files are files with minimal encoding. Word and other word processors create encoded documents that are only human readable with particular software so they might be difficult to read in different operating systems and with different versions of the program. The **portability** and **survival** of plain text files is a major benefit: they will open and display the text properly on any computer, now and in the foreseeable future.

Most word processors can save files as plain text. Spreadsheets can be saved in plain text as CSV (comma-separated values) files. LaTeX documents (.tex) are also plain text files.

Formatting Plain Text with Markdown

Although there are many benefits to writing documents in plain text, you will quickly come across some limitations. You may want to emphasize parts of text with italics or with bold words. You may want to include headings or include quotations.

Markdown is a way of adding formatting to a plain text document. Markdown, like the much more complex HTML and LaTeX, is a markup language that expresses information about formatting using plain text. There are many word processors available that can

display and manipulate Markdown documents with the correct formatting. This guide is actually written in Markdown!

Getting Started with Markdown and Sustainable Authorship in Plain Text using Pandoc and Markdown explain how to use Markdown in combination with Pandoc to use plain text for your academic writing.

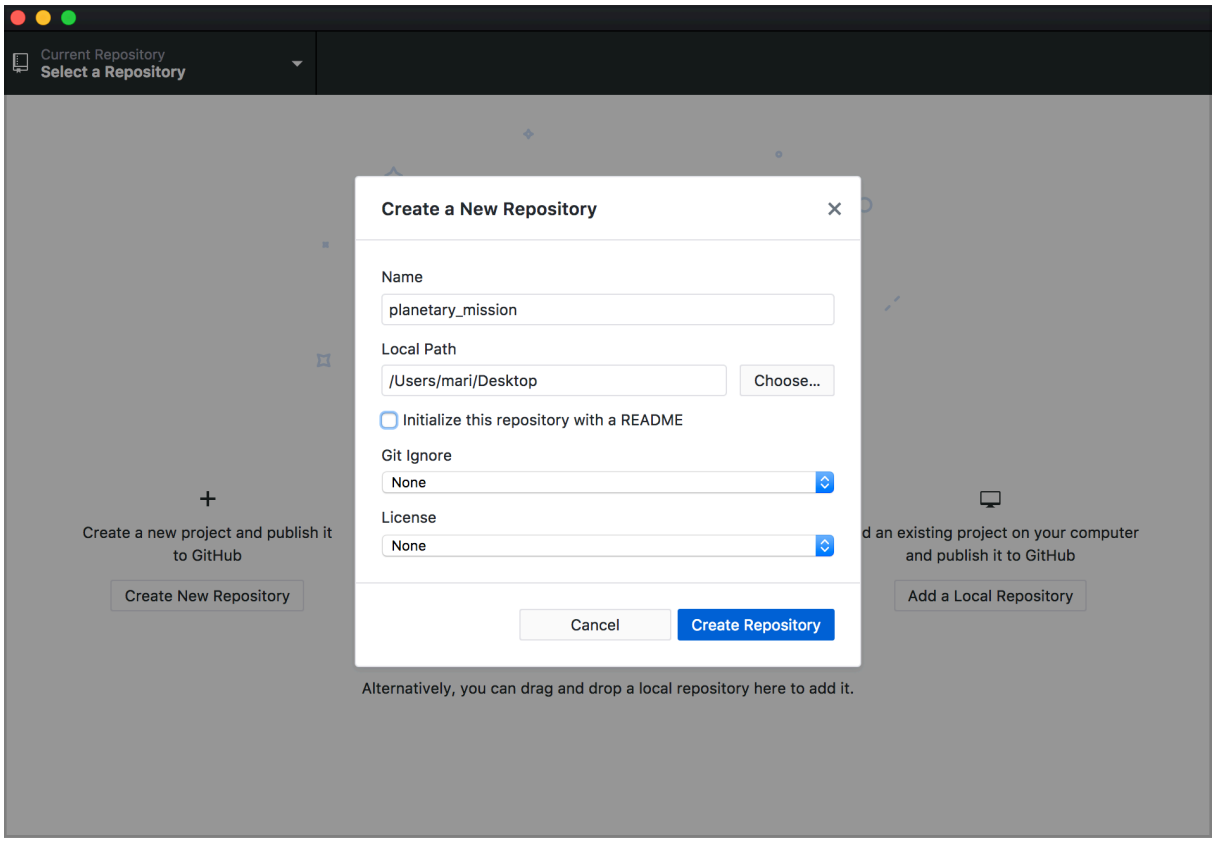
---

1. Creating your (local) repository

A Git **repository** (or **repo**) is the collection the files in a folder that are "watched" for changes by Git. You can have many different Git repositories on your computer. It is best to create one repository for each major project you are working on, i.e., one repository for an article, one for a book, and one for the code you are developing for a particular application.

The folder of a Git repository includes a hidden folder called `.git` (the dot at the start of the name makes it invisible to the Finder). The hidden folder contains the history of changes made to the files that are being tracked. Don't mess with this hidden folder!

We are making a repository for our latest project about terraforming the Solar System. Open GitHub Desktop and select "Create a new project". Give the folder a name (this is the name of the repository) and select a location for the folder in your computer. It's good practice to give names to files and folders that don't include spaces or special symbols.



Clocking "Create Repository" will create a new folder within your computer. If you didn't check the box for a README or either of the pull-down menus, the new folder should look empty. You probably can't see that it contains the hidden folder `.git`, which makes it a Git repository.

2. Adding files to your (local) repo

Start building your repository by saving a simple plain text document to that folder. Use a word processor like TextEdit on Mac or Notepad on Windows to create it. If you *must*, you can create a plain text document with Word by saving the file as Plain Text. Write some text about your favorite celestial body:

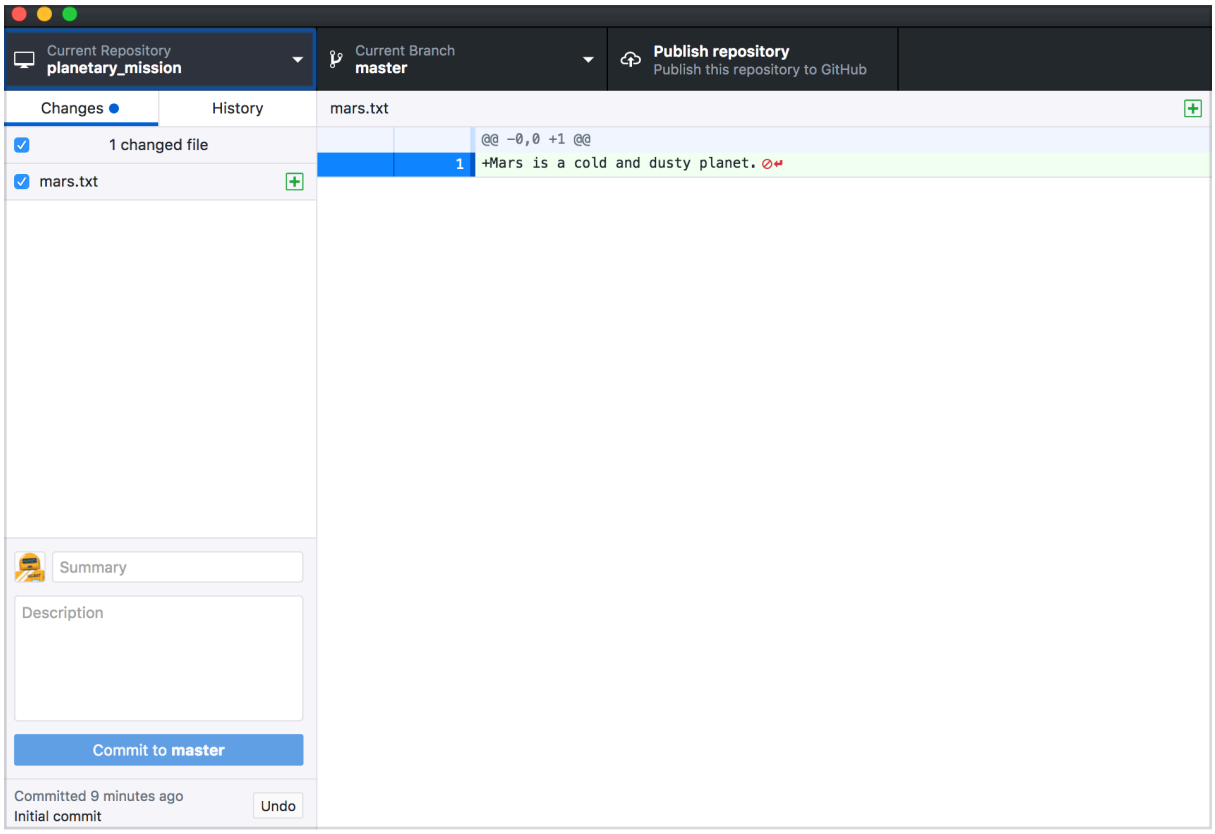
```
Mars is a cold and dusty planet.
```

Save the file to your new folder with the file extension `.txt`. File extensions don't change the encoding of the file (a `.docx` file won't become plain text if you give it the extension `.txt`!) but they help the operating system select the correct application to use.

Make sure that the file is saved in plain text format! Some text editors default to Rich Text Format. You should be able to change this in the Format menu or while saving the file.

Go back to GitHub Desktop. The text file now appears on the left column, in the tab

"Changes". You can also see the contents of the file on the right side of the screen. The first (and only) line of text is highlighted green and has a plus sign next to it because it is a new addition to the file.



Go back to the file in a text editor and make some changes to the text:

```
Mars is now a cold and dusty planet
but it was once a tropical paradise!
```

Save the changes and go back to GitHub Desktop. Notice that it shows all the text as a new addition and doesn't remember the first version of the file. While this text file is inside the repository folder, the file is **not yet being tracked** and the changes haven't been recorded as an official 'snapshot'. To do this we need to **commit** our changes.

3. Committing Changes

A **commit** tells Git that you made some changes that you want to record. Though a commit seems similar to saving a file, there are different goals behind committing changes compared to saving changes. Often you are saving a document merely to prevent losing your progress. Saving the document means you can close the file and return to it in the same state later on. Commits are more like events in history. Each commit records a step in the development of the documents in your repository, and the history of the document

can be traced by looking at the sequence of commits. It's a Goldilocks problem: a repository shouldn't remember every change you made while writing and re-writing one sentence, but it also shouldn't ignore everything that happened between creating the blank file and the paper being published.

Each commit must include a summary and an optional message that describe the changes. This makes it easier to identify what happened at each "snapshot" without having to look through the files. Write a (meaningful!) summary of this commit in the text boxes on the lower left hand side of the screen:

```
First description of Mars environment
```

The blue button below the message fields says "commit to master". This refers to the **master branch** of the repository. Within a Git repository, it is possible to have multiple **branches** or parallel histories of the same files. Different branches are often used to test new ideas or work on a particular feature without disturbing the master branch. **Merging** branches incorporates parallel histories of changes into a single narrative.

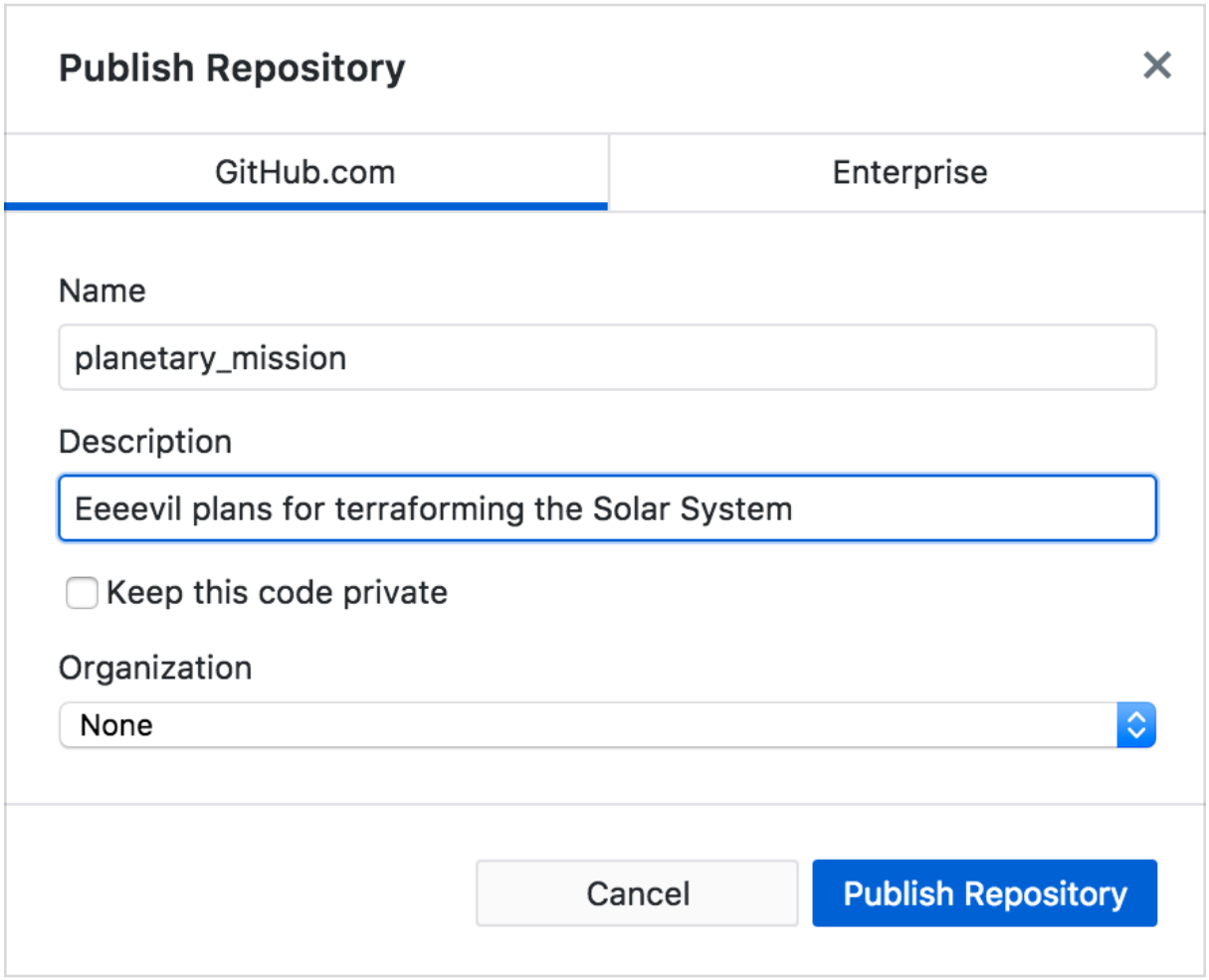
Press the "commit to master" button. The text below the button changed to "Committed just now" showing that you saved a snapshot to your repo. The text file is also no longer listed in the "Changes" tab because there are no differences between the file and the latest version in the repository.

4. Publishing Your Repository

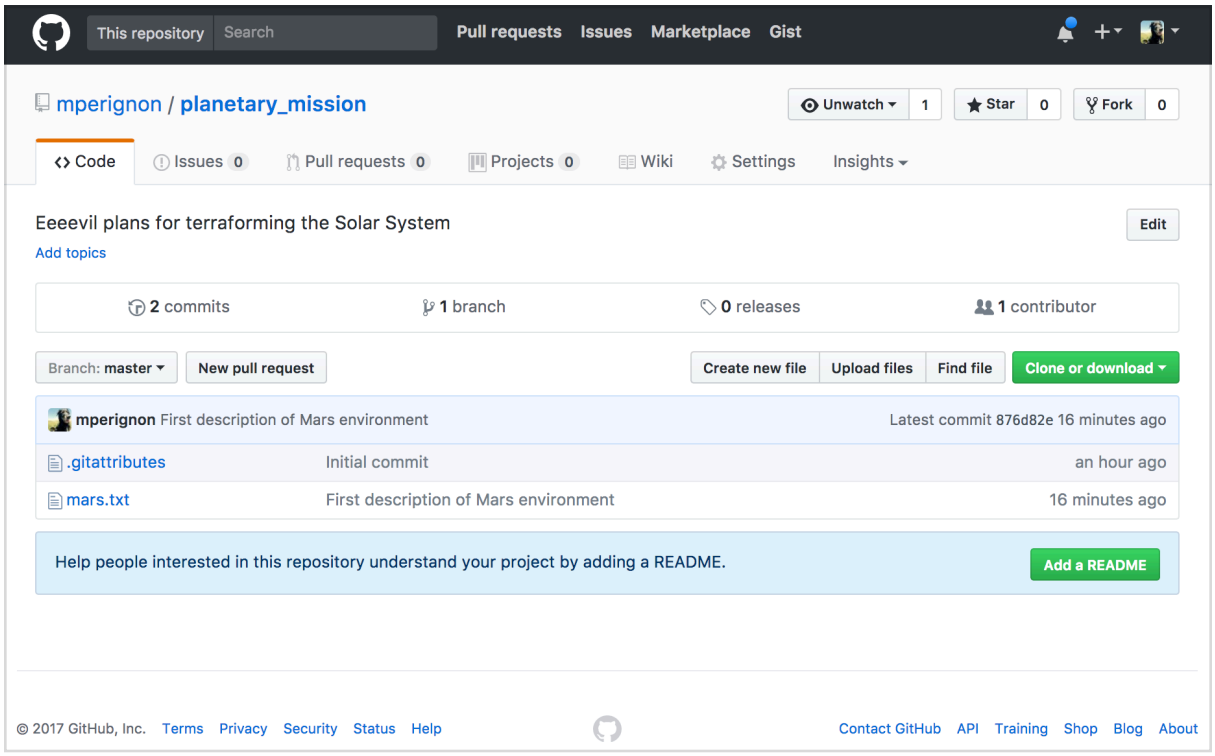
Everything we've done so far has happened locally. While the Git repository is recording the history of the files, they are only save in your own computer. You may be happy to only store the history locally but you may want to upload the repository to GitHub to make it public or to have it stored remotely.

Press the "Publish repository" button on the top bar. This will **push** the history of changes in the **local repository** to a **remote repository** on Github.

Add a short description of the repository and uncheck the "Keep this code private" box. Then press "Publish Repository".



To view your repository online, go to the Repository menu and choose “View on GitHub”. This will open a browser and show your newly created remote repository. Click on the file names to see the latest version of the files. You can also check the history of changes saved in the repo by clicking on “X commits” (the `.gitattributes` file is Github Desktop preferences):



5. Adding More Files

Open the plain text file in your local repository and make some minor changes:

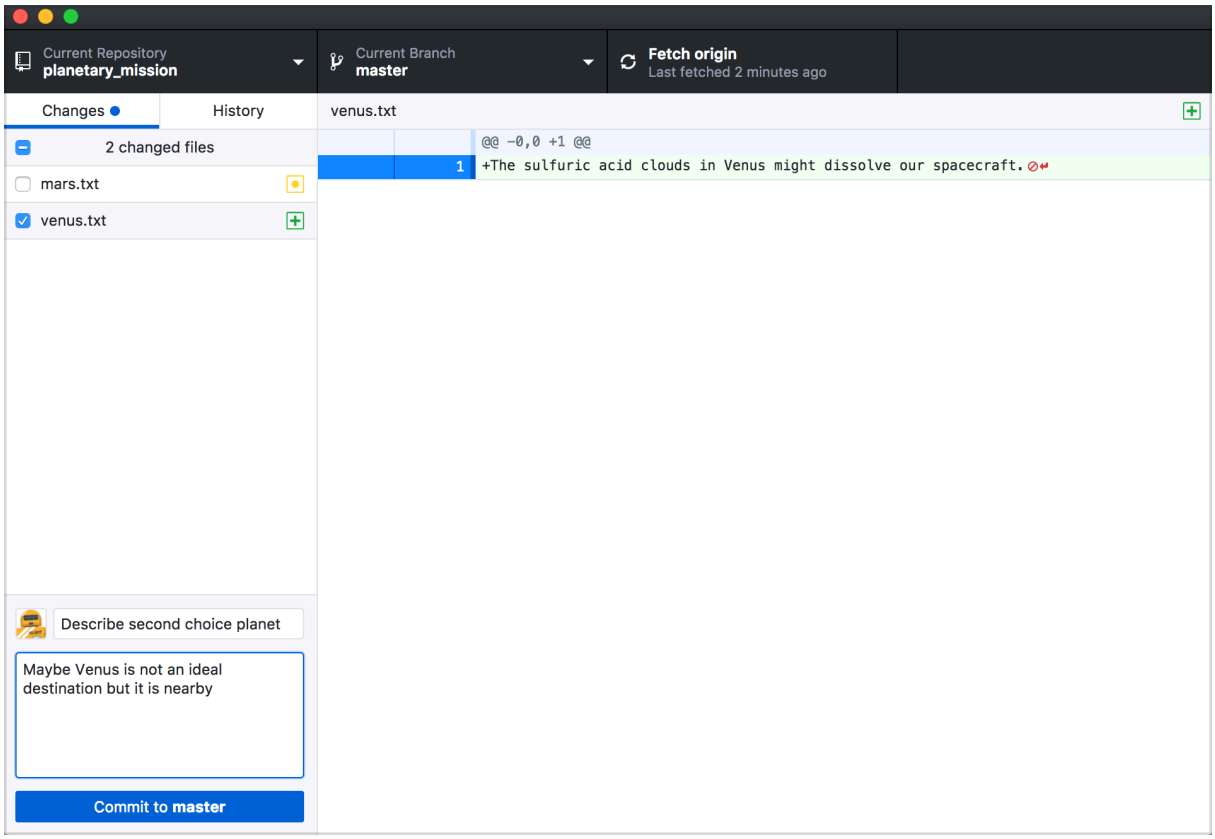
```
Mars is now a cold and dusty planet
but it was once a tropical paradise!

I wonder if little green men get tans…
```

The file once again shows up in the "Changes" tab of Github Desktop because it is different from the last snapshot in the repository. Don't commit these changes yet!

Create a new plain text file in the folder of your local repository with information about a different planet:

```
The sulfuric acid clouds in Venus might dissolve our spacecraft.
```

The "Changes" tab shows both files as having unrecorded changes but we are only interested in committing changes made to the new text file to the repo (we are still working on the first file). Uncheck the box next to the old text file to remove it from this commit. Write a commit summary and press the "commit to master" button:

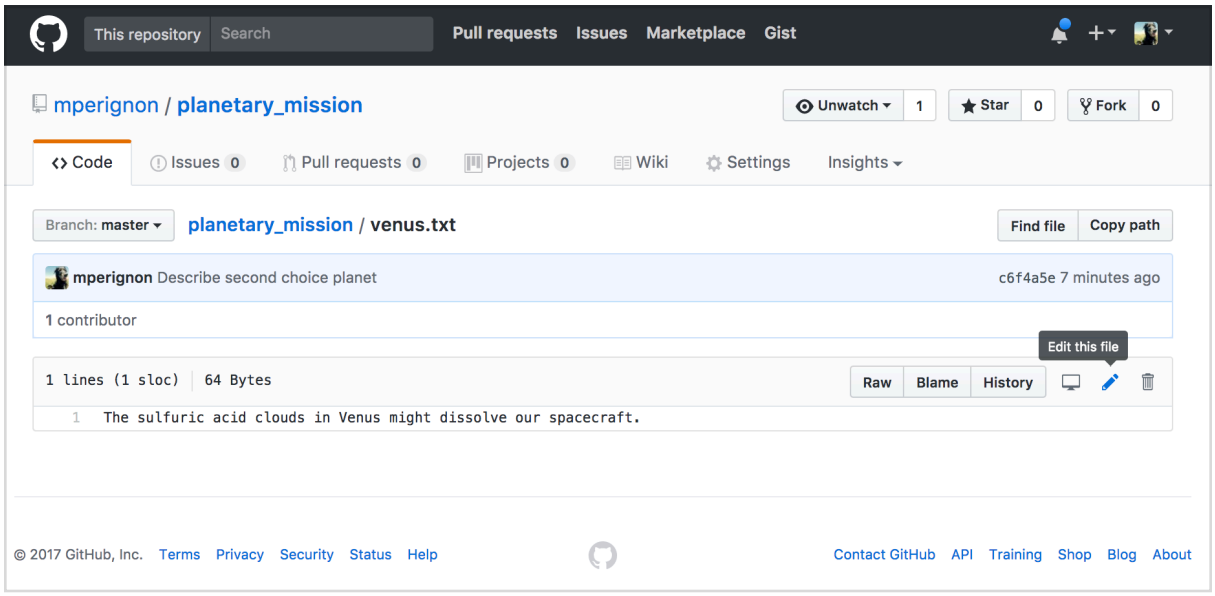


The “Changes” tab still lists the old text file because those changes were not committed. Click on the “History” tab to see the commit history and confirm that we only saved one file. Also check the remote repo (on Github.com) – since you haven’t **pushed** (published) the latest commit, nothing should have changed.

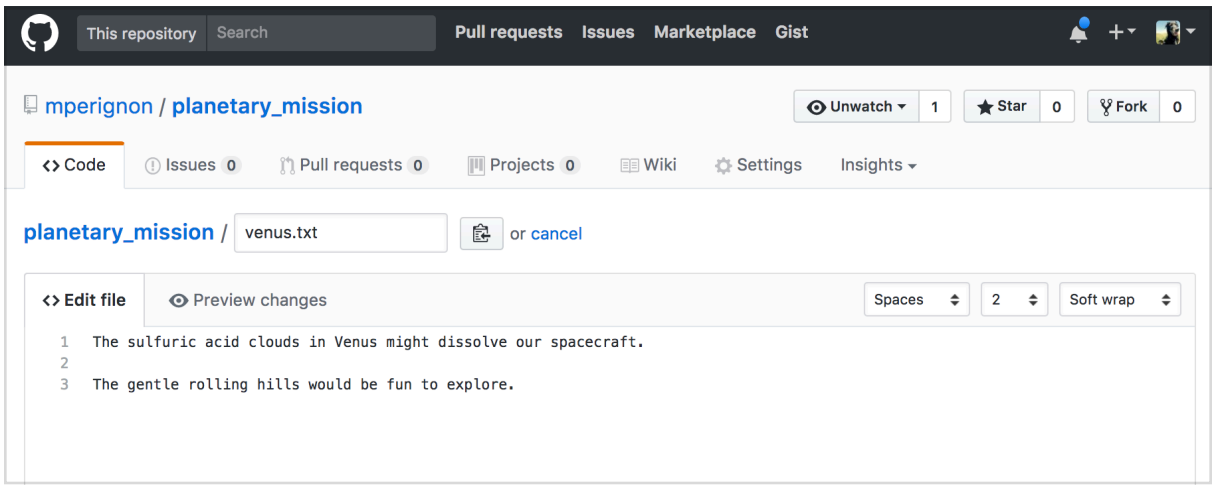
Commit the changes to the old text file to your local repo. Then press the “push to origin” button on the top bar to publish your changes on Github (“Origin” is the default name for the remote repository).

6. Making Changes Remotely

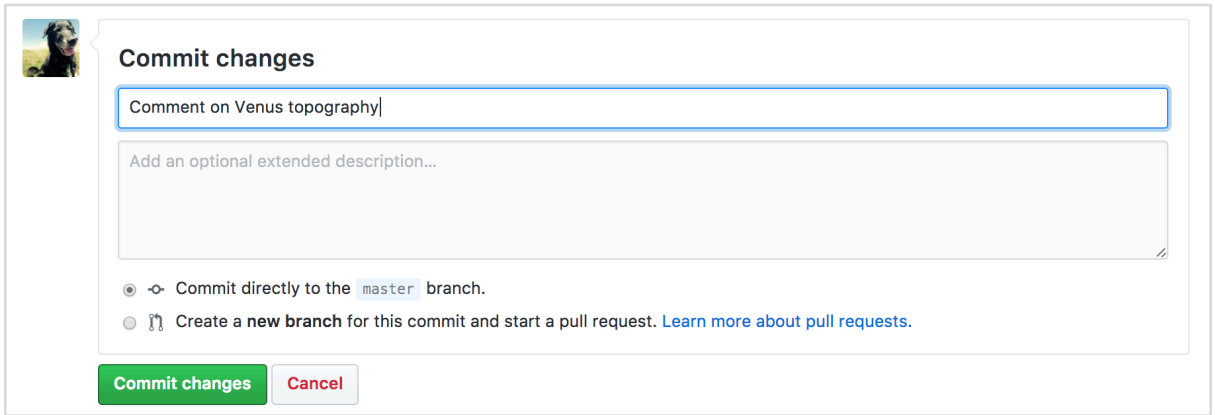
It is also possible to make a change to your repository on the web interface of Github. Look at your remote repository on Github and click on the name of one of your text files to see the contents. Click on the edit option (the pencil):



Add a new line of text to your file:



At the bottom of the page you can commit these changes to the remote repository. Write a summary and commit your changes:



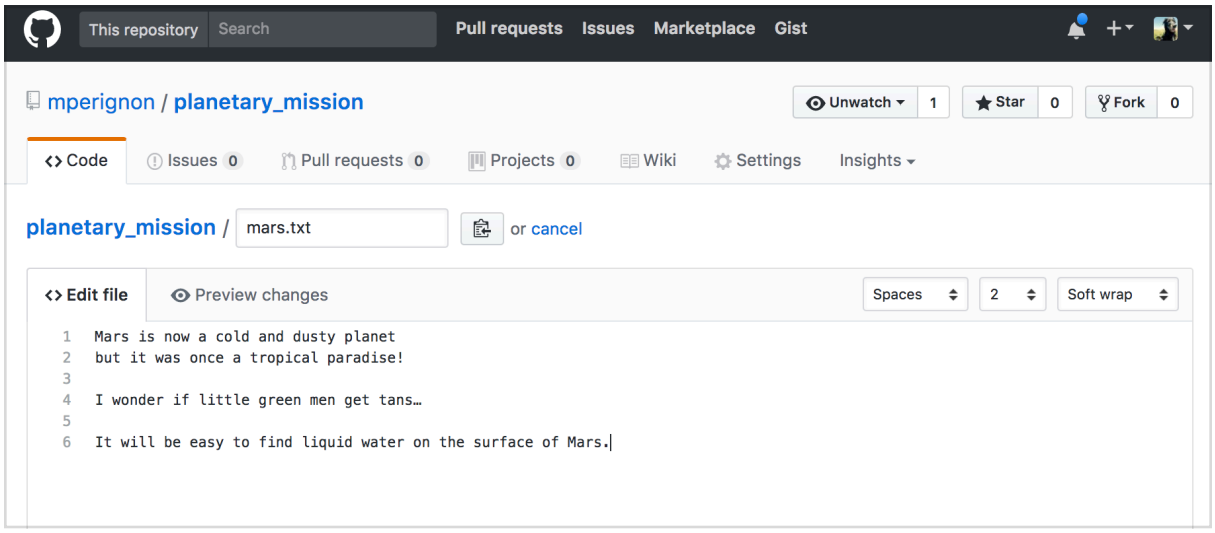
These changes are now stored in the remote repository but *not* the local repository. Go back to Github Desktop and press the "Fetch Origin" button on the top bar to bring any changes from the remote repo back to your computer. Now look at the "History" tab and at

the text files to see the changes.

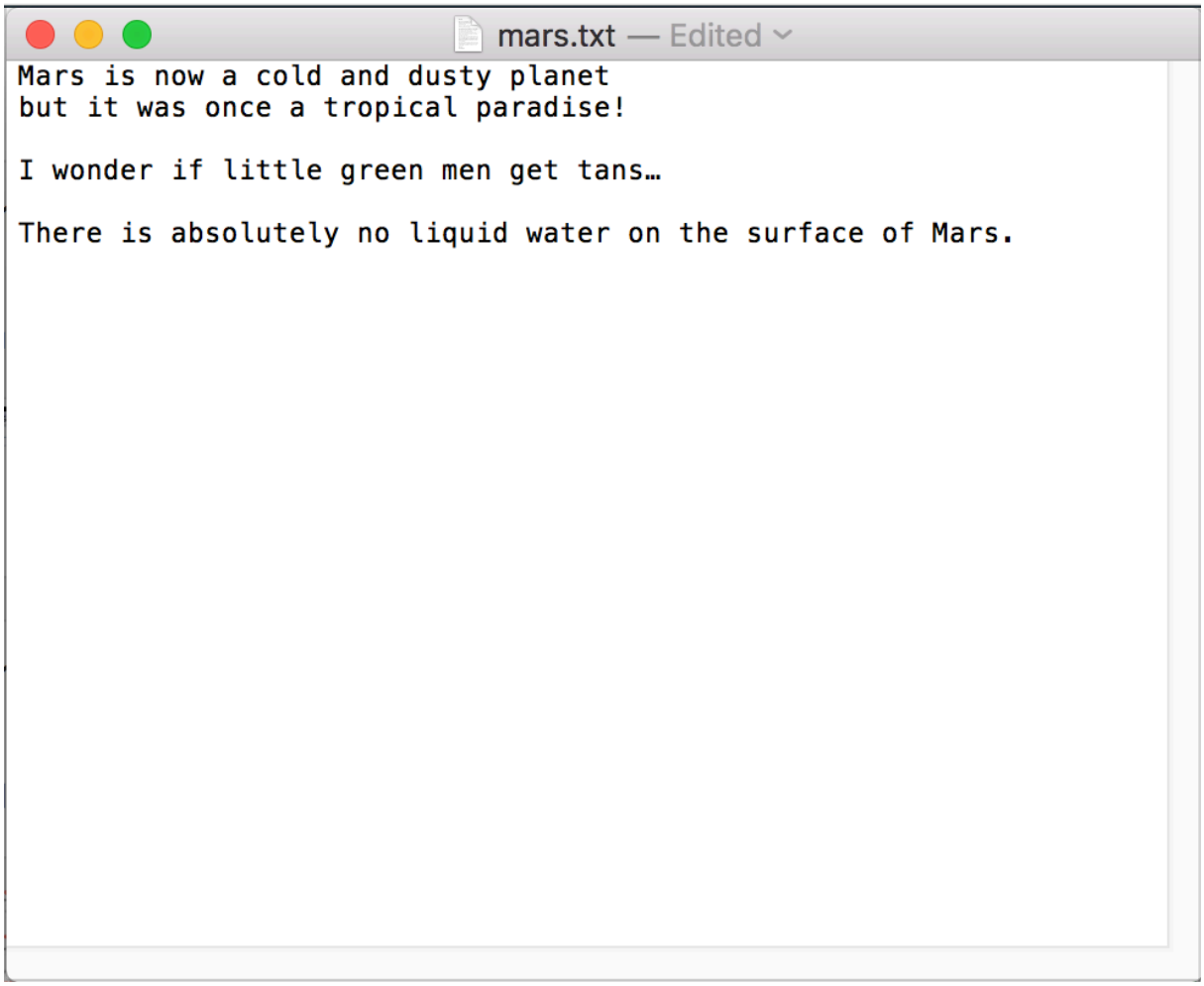
7. Managing Conflicts

A **conflict** emerges when you try to merge two versions of a file with changes that disagree with each other. It is unlikely that you'll run into this problem if you are working alone in one repository, but it can pop up if you are collaborating on one file with someone else. The most likely way a conflict will emerge is if you make a change remotely (on the GitHub website), and then make a change on your local machine without first synching the changes from the website. If you make changes in different parts of a document these changes can be **merged** together without any conflict. Conflicts emerge when changes contradict with one another (i.e. if you try and change the same line of the document in two different ways). Thankfully, conflicts (of the Git kind!) can be resolved fairly easily.

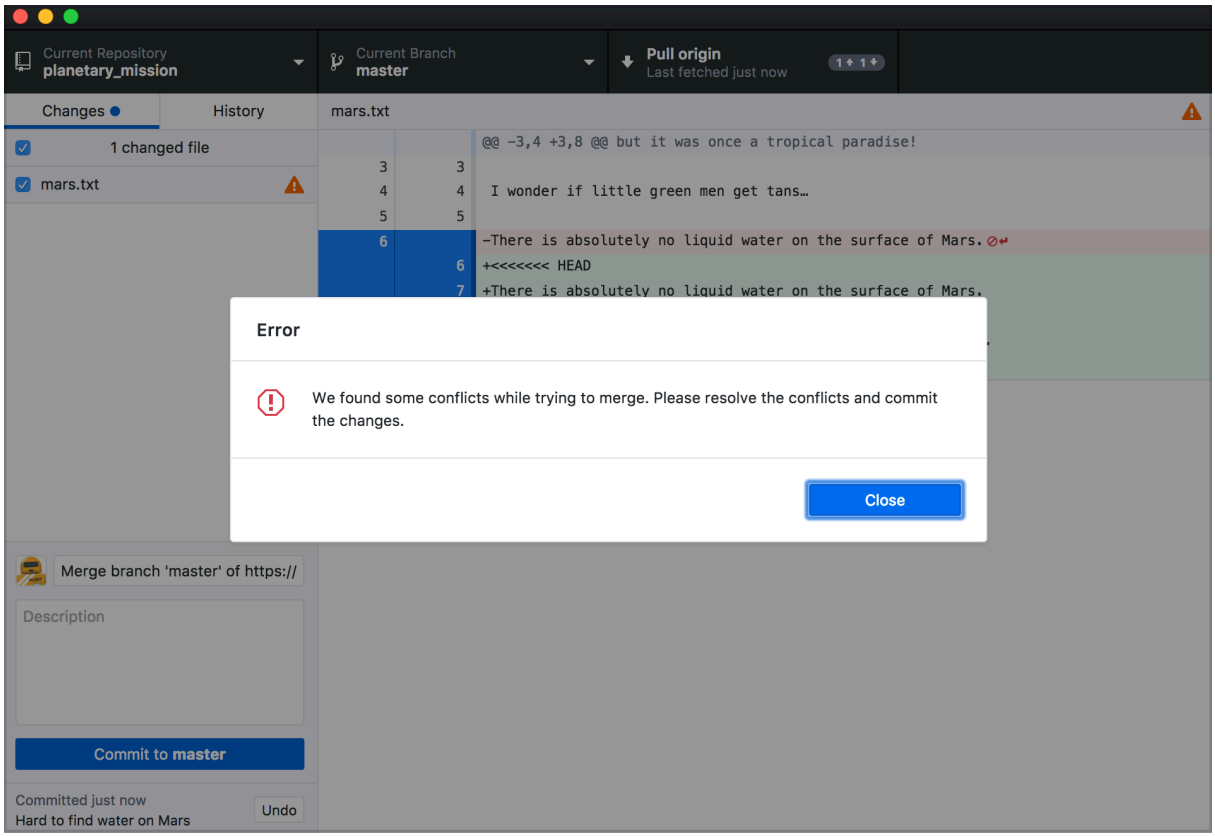
Add a new line to one of the text files in the remote repository (on Github.com) and commit the change:



Don't sync the remote and local repos yet! Open the same text file locally and add a new (different) line. Save the file and commit the change to the local repo:



Sync the remote and local repos (with the “Pull origin” button on the top bar). Since the changes made remotely and the ones made locally are different, it shows that there is a conflict between the two versions of the file:



Open the file in a text editor. Git added some markers to highlight the conflict section:

The conflicting section starts with `<<<<<<<` and ends with `>>>>>>>`. These are known as **conflict markers**. The two conflicting blocks are divided by a `=======` line. `HEAD` refers to the changes made in your local repository, while the long string of numbers after the last conflict marker is the ID of the particular conflicting commit on the remote repository.

There are a number of approaches to dealing with a conflict. You could choose to keep with either of the changes by deleting the version you no longer want and removing the conflict markers. You could also decide to change the section entirely. You can then save, commit, and push the changes as usual. GitHub Desktop will automatically fill in the summary to specify that the commit is to merge a conflict. This is useful if you later want to go back and review the history of the file.

Exploring Other Repos

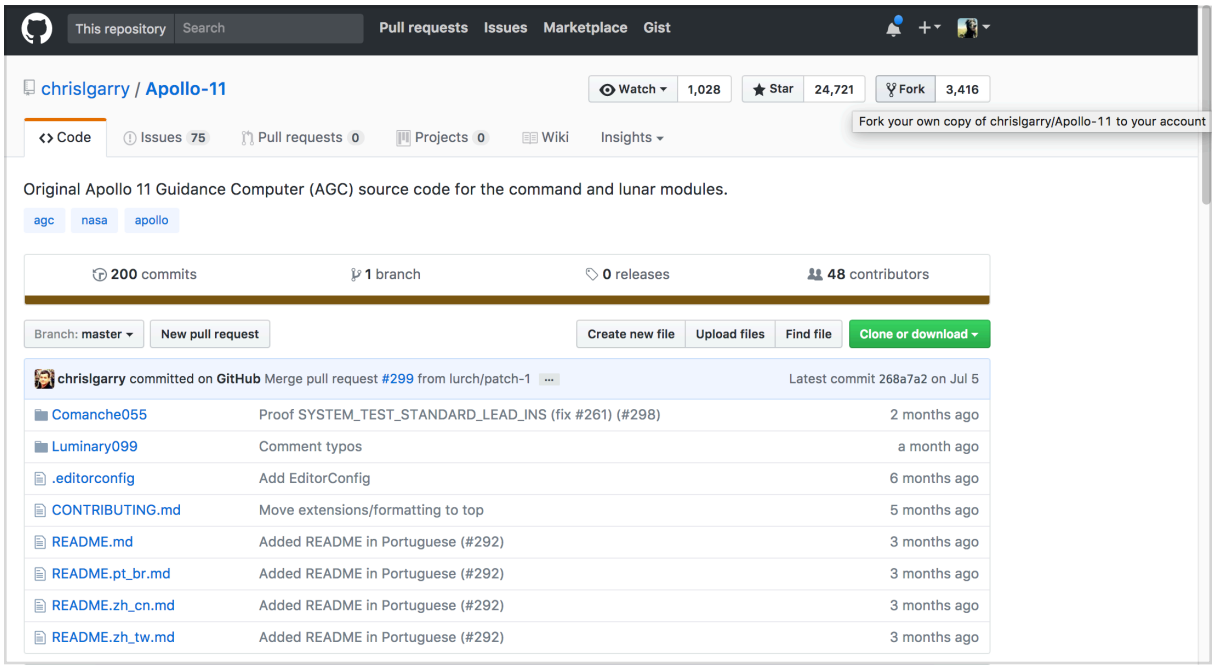
After using GitHub by yourself for a while, you may find yourself wanting to contribute to someone else's project. Or maybe you'd like to use someone's project as the starting point for your own. Or maybe you just think it would be cool to play with someone else's code. This process is known as **forking**.

Creating a "fork" is producing a personal copy of someone else's Github repository. Forks act as a sort of bridge between the original repository and your personal copy. From a fork, you can submit **Pull Requests** to help make other people's projects better by offering your changes to the original project (we will not do that today).

1. Fork a Repository

The original Apollo 11 guidance computer source code for the Command Module and Lunar Module were recently digitalized and added to Github: https://github.com/chrislgarry/Apollo-11

Fork this Github repository to your account by clicking on the "Fork" button on the upper right:

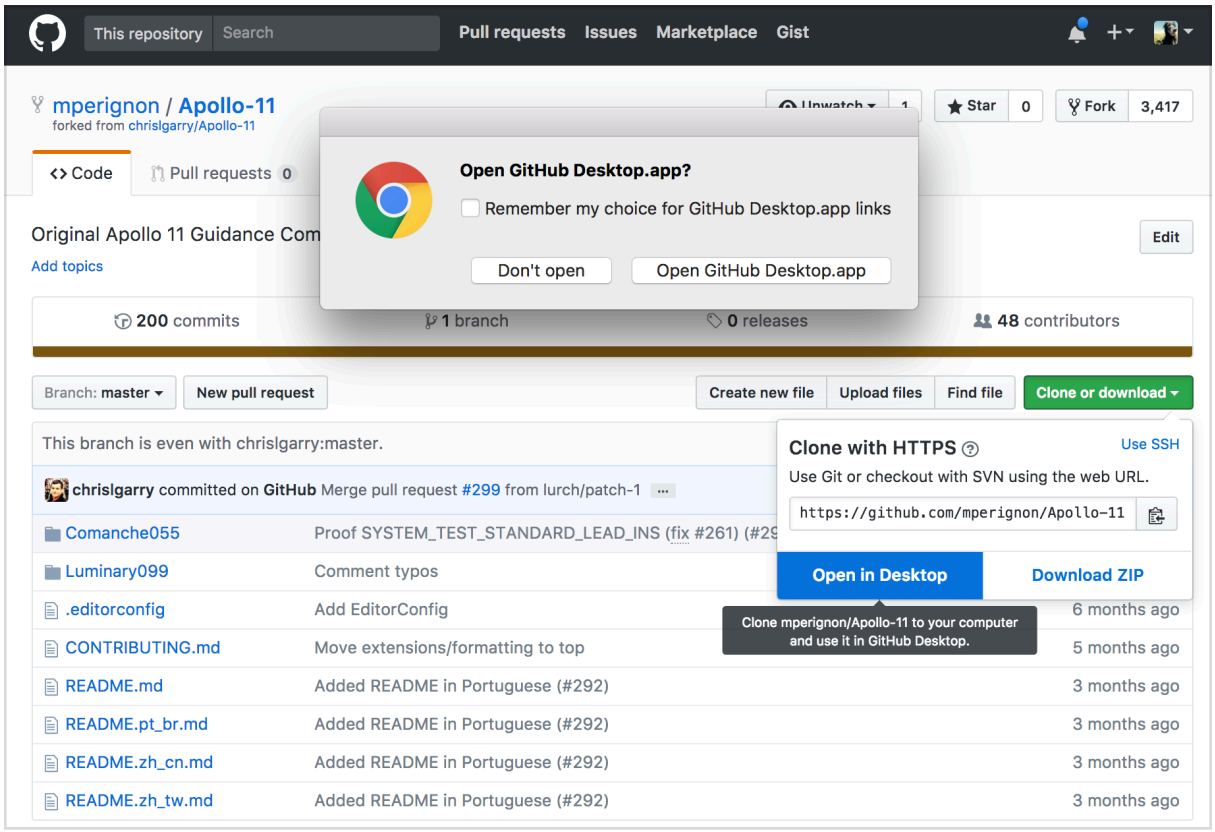


Forking will create a copy of their repo in your own account. At this point, it will look the same *except* that the repo name refers back to its source (the original of a forked repository is often referred to as **upstream**).

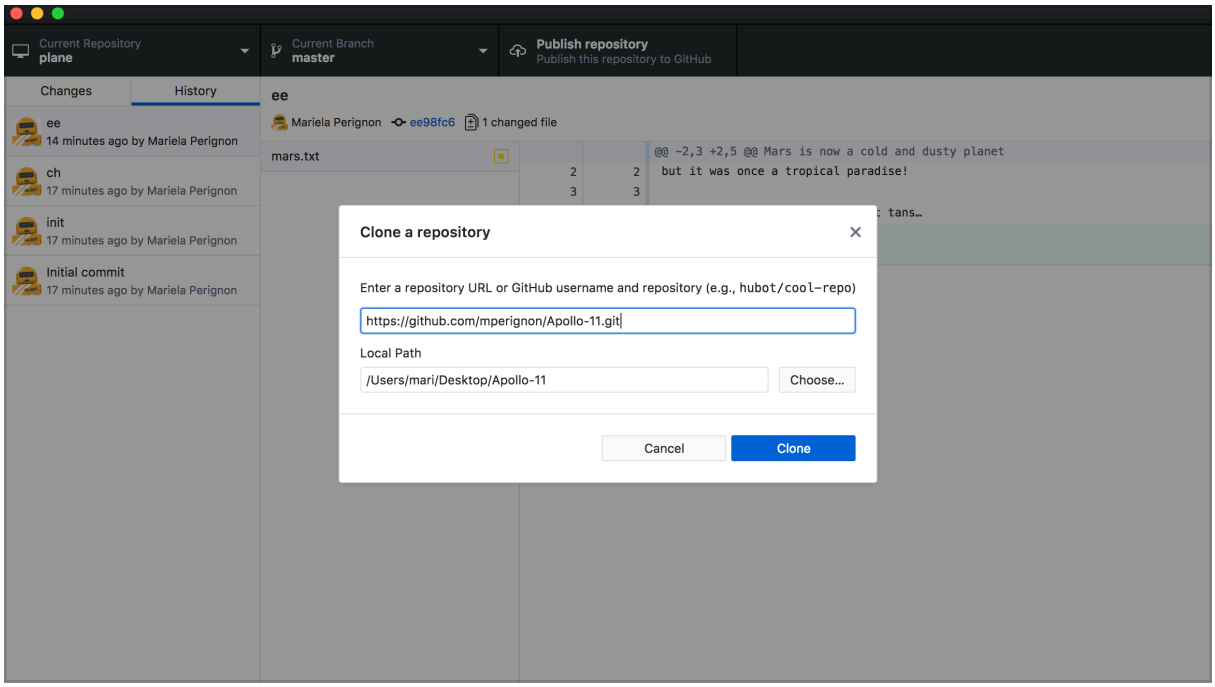
2. Clone a Repository

You've successfully forked the Apollo-11 repository to your Github account. To be able to work on the project, you will need to **clone** it to your computer.

Click on the green "Clone or download" button and then on "Open on Desktop". It will ask if you want to launch Github Desktop:



Then it will ask you where you want to save it:



After you click “Clone”, it will download the contents of the remote repo to your computer. You can interact with it just like you did with the repo we created earlier. You can switch between your repositories in Github Desktop by clicking on “Current Repository” on the top bar.

Exercise:

The terminology and workflow of Git and Github can be convoluted. The only way to get comfortable with these tools is to practice. Repeat the process of creating a new local repository, adding files, and publishing it on Github. You can move existing files into the folder of your new repo instead of creating new ones.